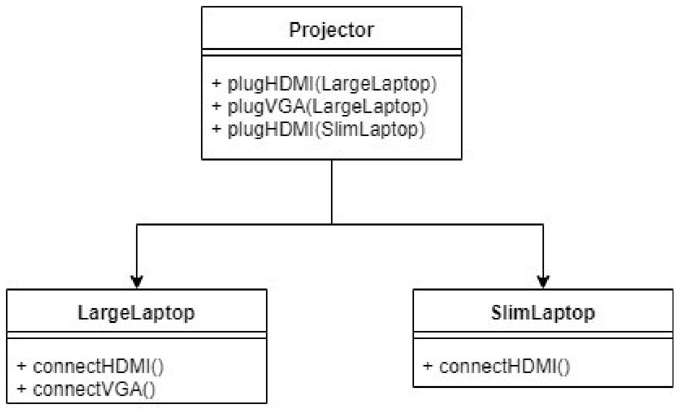
Dependency Inversion Principle

CSCI 2300

Class Dependency

- Class A depends on class B if changes in class B may cause changes in class A.
- Example:
  - Model has `public void battle(int x, int y)`
  - Controller calls Model's battle method
  - Controller depends on Model
  - If we change the signature of Model's battle method, we need to make a change in controller



```
public class Projector
{
   public void plugHDMI(LargeLaptop machine)
   {
       machine.connectHDMI();
   }
}
```

A. Projector calls LargeLaptop's method and Projector depends on LargeLaptop's interface
B. Projector calls LargeLaptop's method, but does not depend on LargeLaptop's interface
C. LargeLaptop calls Projector's metehod and depends on Projector's interface
D. LargeLaptop calls Projector's method and does not depend on Projector's inerface

Identify the dependencies in this design

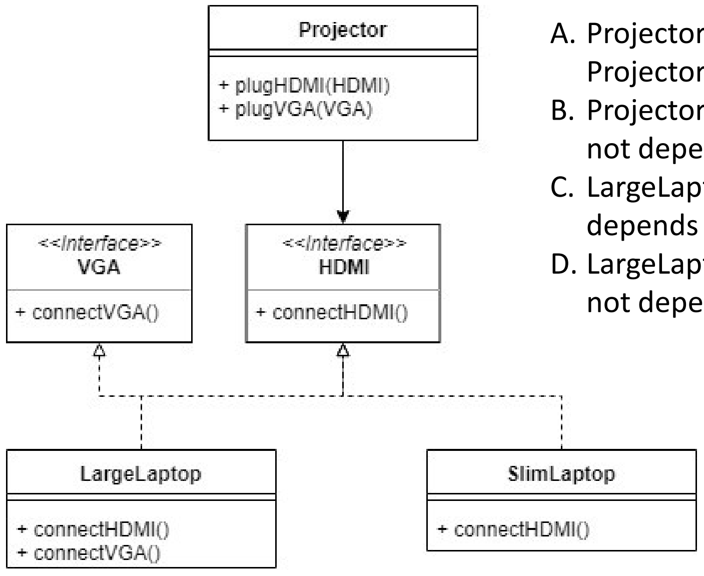
Projector Example Explained

- Projector: higher level module – it calls methods of LargeLaptop (and Slim Laptop)
- Projector depends on lower level modules
- If lower level modules change, Projector needs to change

Dependency Inversion Principle (DIP)

- High level modules should not depend on low level modules
- Both should depend on abstraction
- Abstractions should not depend on details
- Details should depend upon abstraction

`LargeLaptop` object is passed to Projector's `plugHDMI()` method. Identify the dependencies.

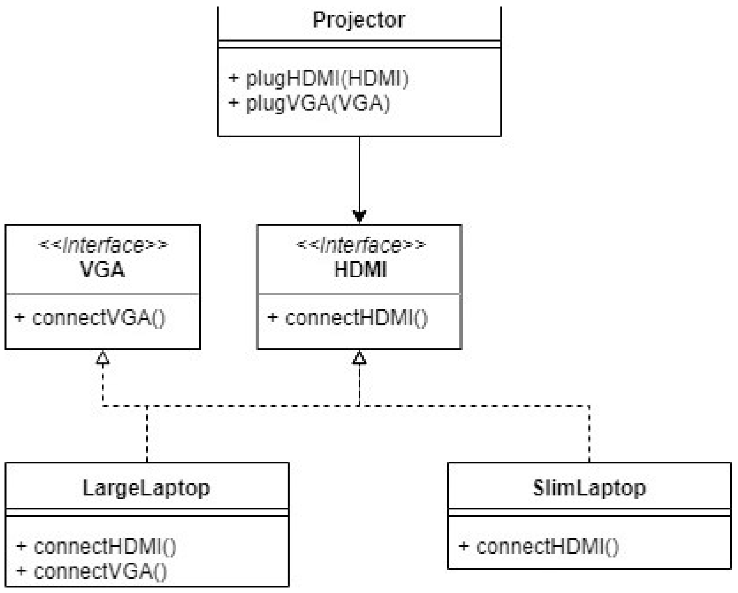


A. Projector calls LargeLaptop's method and Projector depends on LargeLaptop's interface
B. Projector calls LargeLaptop's method, but does not depend on LargeLaptop's interface
C. LargeLaptop calls Projector's metehod and depends on Projector's interface
D. LargeLaptop calls Projector's method and does not depend on Projector's inerface

```
public class Projector
{
   public void plugHDMI(HDMI m)
   {
       m.connectHDMI();
   }
}
```

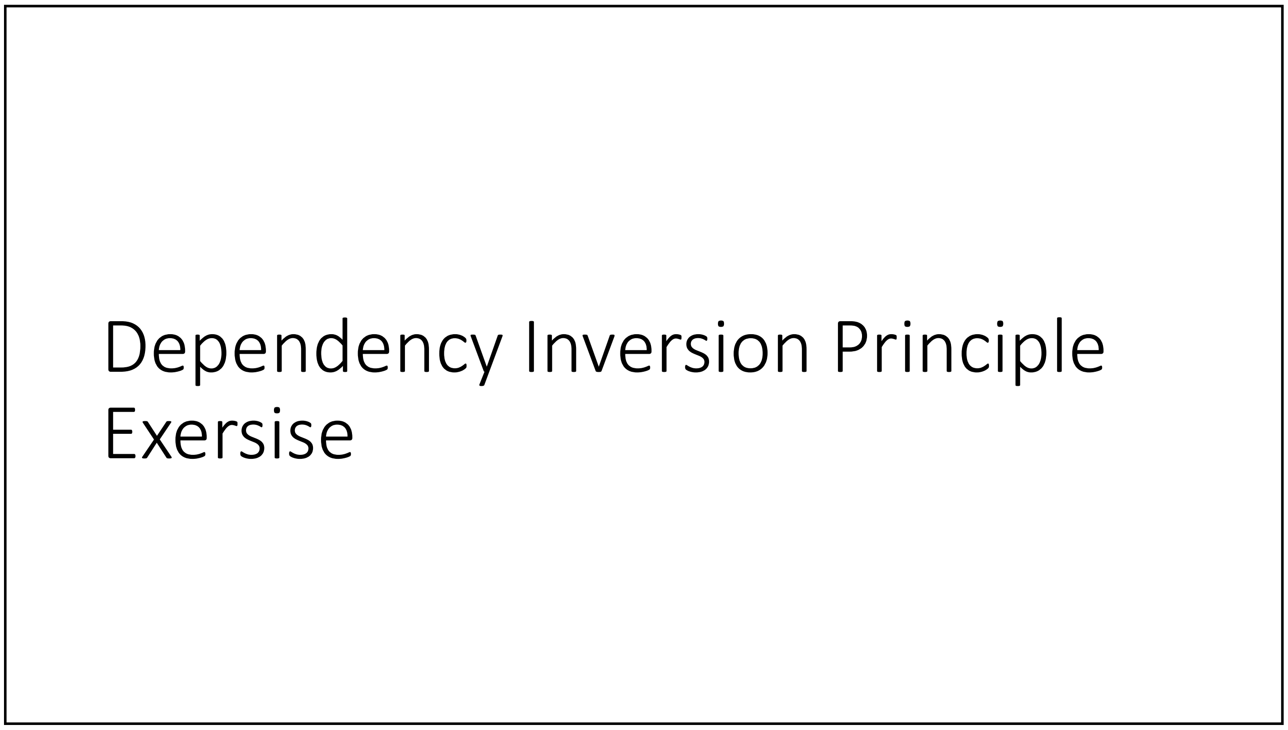
Benefits

- Changes to lower level modules do not impact higher level modules
- New lower level modules can be easily added:
  - How can we add another HDMI device to this design (a Desktop class, for example)?



S.O.L.I.D Design Principles

- S – Single Responsibility Principle
- O – Open/Closed Principle
- L – Liskov Substitution Principle
- I – Interface Segregation Principle
- D – Dependency Inversion Principle



Dependency Inversion Principle Exersise